Q 1. [Referenced CLRS chapter 11]

Load factor is given by

$$\alpha = \frac{n}{m}$$

where n = no. of keys and m = total table slots.

For open-address hash table, we store all the keys within the table itself. $\therefore$ Load factor always has to be ≤ 1.

$$\therefore \alpha \leq 1$$

$$\therefore n \leq m$$

Now, probability that the first slot checked during search or insertion, is occupied will also be given by $\frac{n}{m}$.

~~for~~ The probability that the i^{th} hop or probe to an occupied slot given that first $i-1$ slots were occupied too will be given by $\frac{n-i+1}{m-i+1}$

(This is because we will be searching for an empty slot or the ~~re~~ key we are searching for in the remaining $n - (i-1)$ slots from the $m-(i-1)$ unexamined slots.

Since $n<m$,

$$\frac{n-i}{m-i} \leq \frac{n}{m} \quad \text{for all } i$$

such that $0 \leq i < m$

Let X be the random variable defining number of hops made in an unsuccessful operation

$\therefore$ For all i $1 \leq i \leq m$,

$$P(X \geq i) = \frac{n}{m} \cdot \frac{n-1}{m-1} \cdot \frac{n-2}{m-2} \cdots \frac{n-i+2}{m-i+2}$$

$$\leq \left(\frac{n}{m}\right)^{i-1}$$

$$= \alpha^{i-1}$$

For $i > m$, $P(X \geq i) = 0$.

$$E(X) = \sum_{i=1}^{\infty} P(X \geq i)$$

$$= \sum_{1}^{m} P(X \geq i) + \sum_{i > m} P(X \geq i)$$

$$= \sum_{1}^{m} \alpha^{i-1} + 0$$

$$\approx \sum_{i}^{\infty} \alpha^{i}$$

$$E(X) = \frac{1}{1-\alpha}.$$

As α approaches 1, the expected cost of performing operations on an open address hash table approaches infinity.

However, the expected time is always constant if α is constant

The cost cannot be a constant because in dynamic tables when the hash table is full/nearly full, insert operation can include moving the current keys into the hash table into a larger table. This time will be dependant on the number of values/keys already stored in the table

Assignment 7

Q2

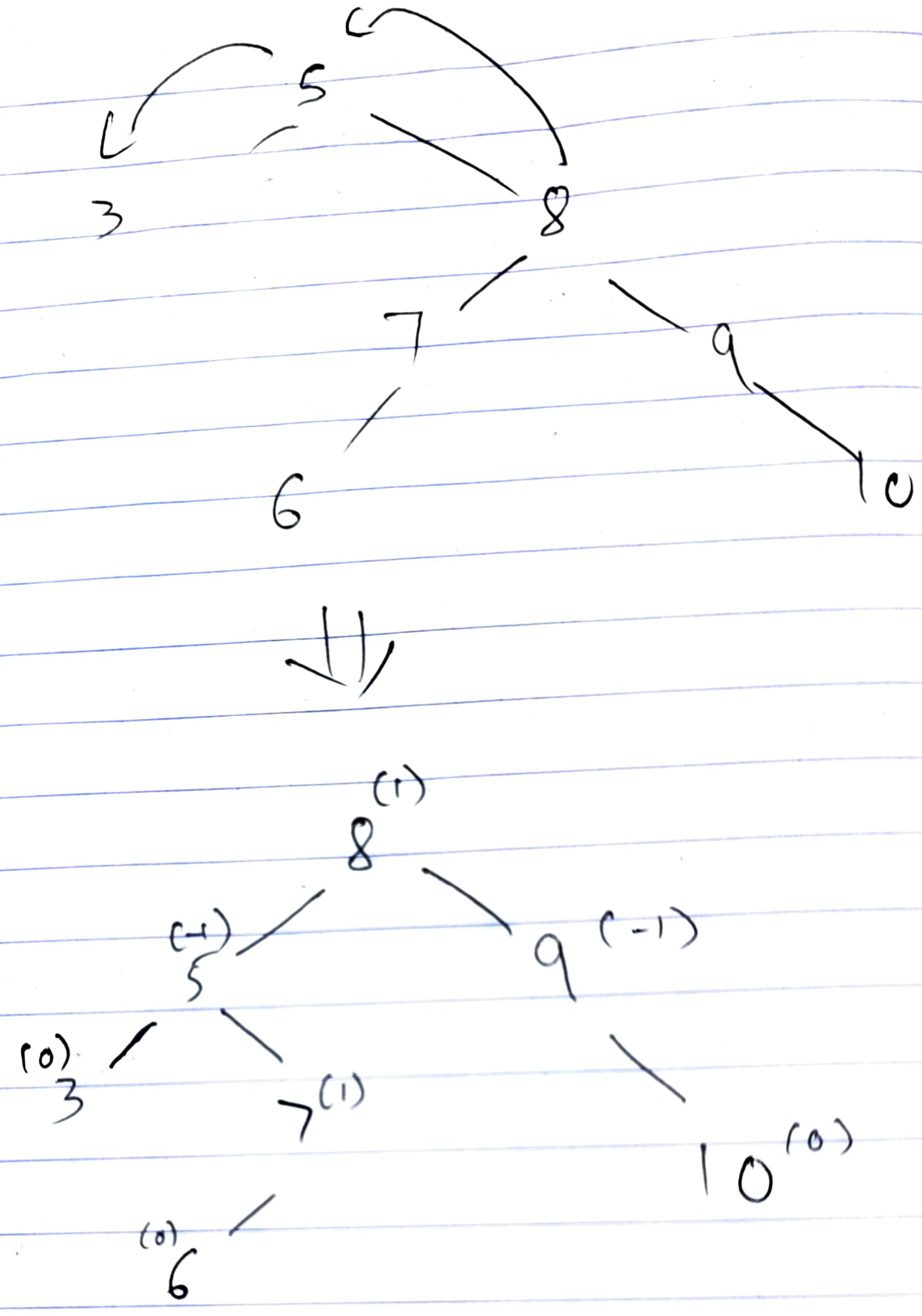
```
RIGHT-ROTATE(T, y)
{
    x = y.left
    y.left = x.right
    if  x.right != T.nil
            x.right.p = y
    x.p = y.p
    if  y.p = T.nil
        T.root = x
    else if  y == y.p.left
        y.p.left = x
    x.right = y
    y.p = x
}
```

where .p → parent
.left → left subtree
.right → right subtree

Q3.

h(k)	index	keys
0 mod 9	0	
1 mod 9	1	10 → 19 → 28
2 mod 9	2	20
3 mod 9	3	12
4 mod 9	4	
5 mod 9	5	5
6 mod 9	6	33 → 15
7 mod 9	7	
8 mod 9	8	17

Q4]



This tree is balanced